International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Pulsed-latch-based register file architecture for multiport

Fazal Shah [1,*], Qureshi Muhammad Kashif [2], Maryam Tariq [1], Ahmad Ali Khan [3] and Bakala Mboungou Marcel Merimee [1]

[1] *School of Computer Science and Engineering, Anhui University of Science and Technology, Huainan, China.*
[2] *School of Mathematics and Big Data, Anhui University of Science and Technology, Huainan, China.*
[3] *Department of Electronics, Quaid-e-Azam University of Islamabad, Islamabad, Pakistan.*

International Journal of Science and Research Archive, 2025, 14(03), 621-642

Publication history: Received on 02 February 2025; revised on 12 March 20215 accepted on 14 March 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.14.3.0705

Abstract

Many difficulties have confronted the system-on-chip (SoC) industry during the previous ten years. These difficulties will likely become more significant as more and more applications revolve on the Internet of Everything (IoE). These issues include reducing power consumption for greater energy efficiency, overcoming numerous causes of variation to ensure reliable performance, and reducing design area to save money and boost integration. As a result, chip designers encounter various hurdles when attempting to create stable structures with complicated angles of capability while preserving a small die size and power expenditure. Memory components are among the most critical circuit components on every chip. They account for the majority of the chip's size and power consumption, which has an impact on the overall efficacy and reliability of the chip. These consist of a diverse multitude of serial elements in logic paths, caches, register files, and large memory arrays. In contemporary synchronized CMOS circuits, sequence elements are indispensable components. In fact, they can be responsible for up to 50% of the average number of cells in a semiconductor. We propose a novel methodology aimed at enhancing the reliability of pulsed latches while ensuring that there is no substantial decline in efficiency, area, or power consumption. Furthermore, given that sequential elements can be utilized in creating compact register files, the implementation of pulsed latches in register files is reviewed and compared with other conventional implementations, including static random-access memory (SRAM) and flip-flops. Also shown are new implementations of multiport register files, which are highly beneficial for many applications. The suggested approach has been proven to considerably decrease the enormous excess in area, power consumption, and latency that is typically associated with conventional methods of designing multiport register files.

Keywords: Register files; SRAM; Flip-Flops; Pulse generator; Virtual ports

1. Introduction

A significant amount of power in contemporary CPUs is allocated to register files [1]. Along with the cache array, they also take up a lot of space. Consequently, the design of the register files has a significant impact on the performance, power, and reliability of the processor's overall operation. The register file is often the most heavily used part of a chip due to the large volume of reads and writes it receives. As a result, the devices' performance can suffer and can rapidly increase the devices aging [2]. Register files in CMOS circuits have been implemented using SRAMs (static random-access memories). Due to the need to store a substantial quantity of bits, each SRAM cell, which retains a single bit, is engineered to be as compact as possible to facilitate effective read and write operations while maintaining sufficient noise margins. Data may also be stored via standard cells like flip-flops and latches. However, while sequentially cells are developed in the library for various uses inside a chip's primary core, compared to SRAM cells, which are exclusively used for storing data, the design requirements and limitations are much different. A flip-flop or latch maybe three to five times larger than an SRAM cell. As a result, SRAM is recommended for implementing register files of medium to

* Corresponding author: Fazal Shah

Copyright © 2025 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution Liscense 4.0.

large size. Nevertheless, SRAM is thus recommended for constructing medium- to large-sized register files. SRAM is not desirable for compact data storage because the area gain of single cells for a small register file may be less than the space and power overhead of the SRAM peripherals [3]. The use of typical cell-based register files instead of SRAM for smaller store capacities may therefore be more attractive. Additionally, standard cell-based register files are typically preferred for chips that require an ultra-wide range of voltage or use an ultra-low voltage supply [4, 5, 6, 7].

2. Related Work

The implementation of register files has been extensively discussed in the literature. A register file design including several banks was created in [8] to reduce access time and mitigate logical complexity. A level-2 register file layout was proposed in [9] for superscalar processors to decrease both the size and the quantity of necessary ports. Several circuits for register files were assessed in [1]. Their energy efficiency relative to each other and the number of registers and ports was the main point of comparison. To cut down on power usage, an exclusive asymmetrically ported register file implementation was suggested in [10]. This power savings is due to the fact that certain ports are limited to reading and writing to the register's lowest significant value only, rather than the whole register. By increasing the number of read and write ports in register files from two to four, array replication and double pumping were examined in [11]. The use of bank replications of block RAMs and multi-pumping to implement register files on FPGA was investigated in [12]. An innovative design for multi-pumping was proposed that utilizes shift registers to deliver high performance while conserving power and space. In [13, 4], the traditional method of implementing register files using cells, which only have one read/write port, was discussed. There was a comparison and discussion of the SRAM implementation with the use of flip-flops and latches to construct the register files. Pulsed latches were used to create a 16x32-bit register file with one write port and two read ports for an extremely broad voltage range [5]. The space and power requirements are evaluated in comparison to an identical register file built using flip flops. Reference [14] examined the application of STT-RAM for constructing GPU register files. Two methods for improving the implemented register files' performance and power consumption were tested. As a result, it appears that considerable work has been expended in proposing alternative architectures for the efficient implementation of registration files. The schematic level was responsible for implementing certain projects, while the circuitry level was responsible for executing others. The objective was to obtain an efficient register file implementation in terms of energy that performs at the required efficiency level while occupying the least amount of space. Furthermore, innovative methodologies were investigated to incorporate additional read and write ports without substantial performance deterioration and with little size and power overheads.

2.1. Single-read, single-write register file design

We will examine four different ways that a shared register file with read/write ports may be implemented. Therefore, from now on, it will be known as the 1R1W registry file. In addition, we assume a register file of dimensions $W * B$, where W denotes the quantity of word registers and B signifies the amount of bits per word. One way to look at the register file is as an array with B columns and W rows. Additionally, we presume a word access method that enables the entire register to be read and written, as is common with numerous register files. Furthermore, it is assumed that read and write operations have a delay of no more than one clock.

2.2. Implementation Based on SRAM

Many different types of applications have made heavy use of SRAMs, such as register files, caches, and big data storage. Traditional memory arrays frequently use the six-transistor (6T) SRAM bitcell as a building block; however, this kind of memory only supports a single read/write operation per cycle due to its single interface. Certain 6T devices enabled two read operations or one write operation each cycle by dividing the word line in half [15]. Time-sharing is still needed in order to do write and read tasks at the same time. Consequently, the write action occurs in the earlier half of a cycle of the clock, whereas the read action happens in the later half. Moreover, this places restrictions on the minimum clock time and calls for proper timing.

Constructing a register file with autonomous read-write ports increases the complexity of the memory bit cell. Fig 1 illustrates the outcome of a prevalent modification that incorporates an additional two transistors to create an 8T bitcell [16, 17, 18]. As with any regular 6T SRAM, the write operation is executed in a conventional manner. The write row address decoder activates the word line selector (wr), and the column decoder and sensing amplifiers apply the new data to the wr bit and wr bit (the write bit lines). The device reads one end at a time; consequently, the read line selection (rd) must be elevated to link the read bit line (rd bit) to the designated cell. When the cell stores '1' (Q is VDD), the read bit is subtracted from its previous changed value; conversely, when the cell stores '0', the read bit remains at its prior charge value. Reading from a lower supply voltage is more stable because the read transistors separate the storage node when reading [19, 20]. Two additional read transistors are embedded on the side of QB certain other SRAM

bit cells to create a differential read interface [16]. This will lead to a significant increase in area when the SRAM bit cell is upgraded to a 10T cell. Though there are two distinct column decoders for read and write data and two two-row decoders for read and write addresses, the remaining part of the circuit is the same as a standard SRAM. These decoders are in charge of producing the rd and wr signals for the bit cells.

2.3. Implementation Based on Flip-Flops

The SRAM register file and the FF-RF (flip-flop-based register file) vary in a subtle structural way. The read logic circuit, the flip-flop array, and the write logic circuit are the three distinct parts that comprise this circuit, as seen in Fig. 1.

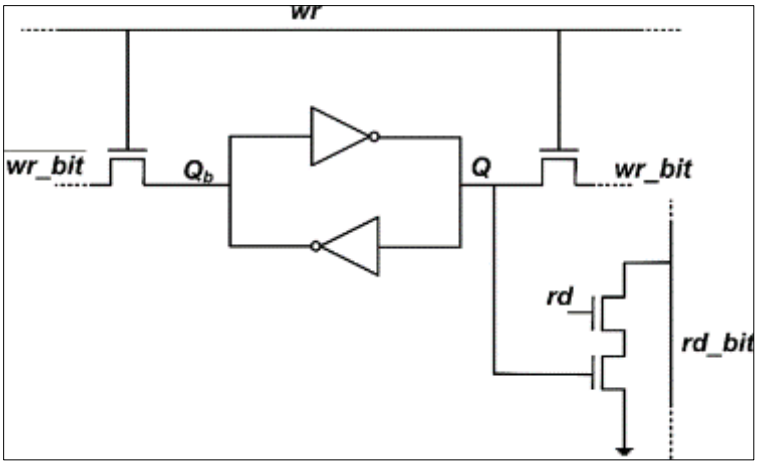

Figure 1 Conventional 8T SRAM bitcell layout for 1R1W register file

2.4. Write Logic

A write operation may be performed on one register of the flip-flop array at the next clock edge thanks to the enable signals generated by the write logic circuit, which is a block in the circuit. The conventional version of this decoding circuit uses the write enable (wr en) and write address (wr address) as inputs and outputs, respectively, as W enable signals.

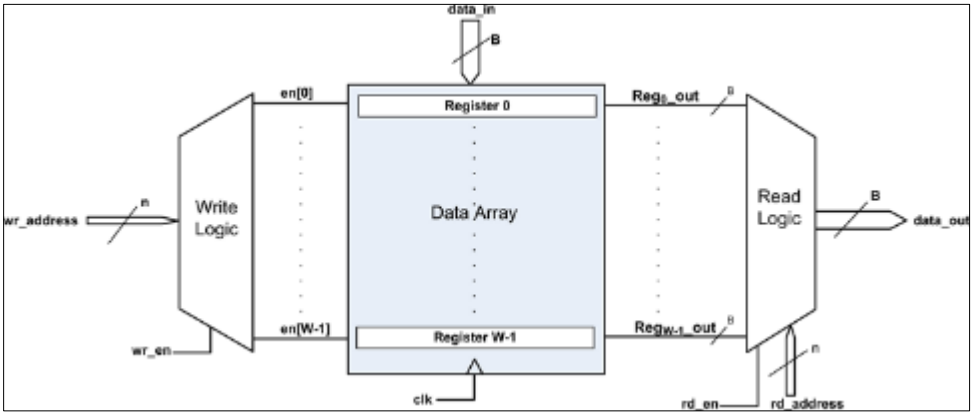

Figure 2 The conventional cell-based register file block diagram

2.5. Array of Data

The data array shows the real parts of the data storage. The flip-flops used to build this data collection are part of a register file that is based on flip-flops. W B flip-flops make up the array. Every B flip-flop is activated by an identical trigger signal generated by the write logic block to operate as a register. To facilitate the writing process, two distinct approaches may be used. Initially, flip-flops using an input enable signal are used. These flip-flops may store fresh data only when the clock edge reoccurs, depending upon the activation of their allowed signal. Otherwise, the stored data will remain unaltered after midnight. A 2-1 multiplexer is often included at the input of a basic flip-flop to facilitate its

operation. The external data enters the multiplexer, and the current output is sent back into it. Once this method is used for generating the data array, the exact same clock signal is going to be sent across the entire flip-flop array.

The second option involves the use of basic flip-flops that do not have an enabling feature. The clock signals of these flip-flops are driven through clock gating cells (CGC), which are shared by each row of the array. These clock-gating cells are the only ones that receive the clock signal and the allowed signal in this case. As seen in fig 3, the outputs of these cells are used to clock the respective flip-flops. Upon selecting a register for reading, the relevant clock gating cell is activated. Thus, the pulse of the next cycle of the clock is sent to the specified row of flip-flops, where the source of write data is sampled and preserved. Simultaneously, the clocks in the other rows are often linked to logic '0' and are non-operational.

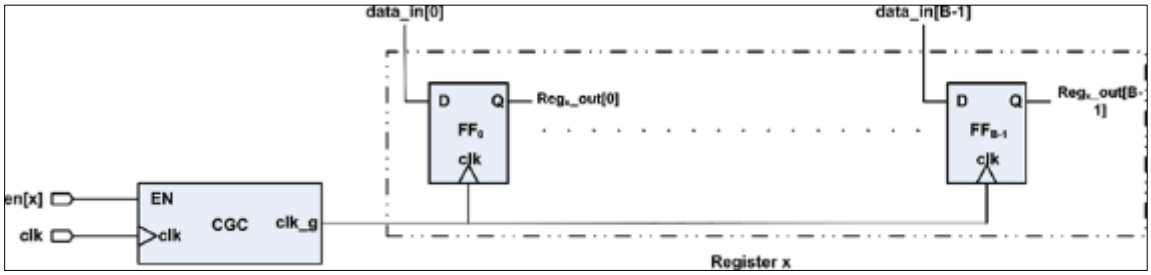

Figure 3 The data array's flip-flop registers showing the clock gating cell (CGC) applies to turn on the flip-flops

We found that the second method saves space and power when we compared the two methods in our research. We use only one clock gating cell for every row of flip-flops instead of greater size-enabled flip-flops, which exceed traditional flip-flops in our technology by over 35% in the area. Using the clock gating cell strategy will significantly reduce area requirements for flip-flops with a number greater than 2 (i.e. B ≥ 3), as the added clock gating cell has an area that is only marginally larger than that of a single flip-flop. Moreover, the clock gating technique directs the primary clock signal towards the clock gating cells instead of the whole flip-flop array. By using the clock-gating strategy, a considerable amount of power is conserved during clock-switching.

2.6. Read the logic

In order to send data to the output port, the read logic circuit retrieves it from one of the array registers. On the basis of the input read address (rd address) and the input read enable (rd en)2, a specific register is chosen. Most register files employ an asynchronous read operation to update the output data instead of waiting for the next clock cycle to finish. Nevertheless, in specific designs, the read address input contains a latching stage—either flip-flops or latches—that sampling the address located at the currently running clock edge at the very beginning of the read cycle. Then, in the register file, the selected read resistor is based on the sampled address. Thus, we can verify that the read address stays constant throughout the read process and prevent any potential issues that may arise from bits not arriving simultaneously. Since other processing core circuitry usually generates read addresses, this can make this logic run more smoothly (it is not necessary to alter all address bits simultaneously) and provide the different read address bits some breathing room in terms of latency.

B parallel W-to-1 multiplexers are often used to construct the read logic block. In most cases, this method works better than using a decoding circuit to turn on 3-state buffers at each register's output. 3-state buses can significantly impair the temporal integrity of selected data when transported over extensive distances due to their buffering difficulties [4].

2.7. Implementation Based on Latches

The layout of the 1R1W register that uses latches opposed to flip-flops is identical to that of the flip-flop that was explained before. The register file that uses latches should be smaller than one that uses flip-flops due to the lower size of the former. Due to decreased leakage and internal power consumption of latches, the version that is based on register files is also expected to utilize less power. Latch timing restrictions are more severe than flip-flop timing constraints for a 50% duty cycle clock signal (half of the clock cycle) due to the latch's wide transparency window. When reading and writing to the same register causes the latch to enter a closed loop, the viewable window of the latch allows for the employment of external circuitry to feed the latch's outcome back into its input. One possible solution is to restrict the amount of data that may be read and written into the system. Another option is to incorporate flip-flops or latches, which are visible during the opposite fifty percent of the clock cycle, within the circuits that utilize this type of feedback [13]. Nevertheless, this will affect the timing of these external logics and add some power and space overhead.

2.8. Implementation of Pulsed Latch

An agreeable compromise between latch-based and flip-flop-based register files is provided by pulse latch (PL)-based register files, allowing for optimal performance in both cases. PL's timing concept is quite similar to flip-flops since they are latches that are driven by brief pulsers. Moreover, they are anticipated to use less energy and occupy a smaller area compared to flip-flop based register files, similar to latch-based register files. When contrasted with the register file that relies on latches, the pulser is the only additional overhead. Considering that every B latch in a register uses the same pulser, this overhead is really little. The clock gating cell may be redundant since the pulser circuit seen in Fig 4 controls the latches' transparency via its produced pulse. Thus, to enable control of pulse creation via write logic, it is necessary to update the pulser circuit. Our proposed layout for the enhanced pulser circuit is shown in Fig 5.

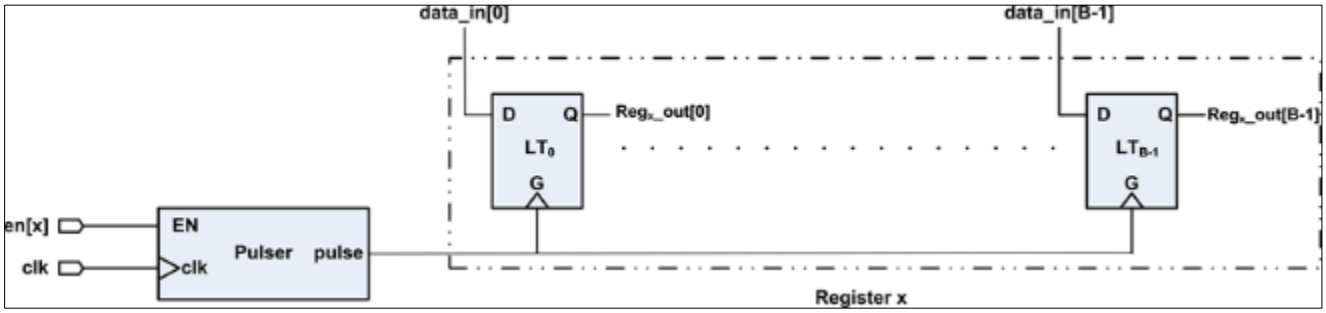

Figure 4 One of the data array's pulsed latch registers shows the pulser circuit has been used instead of the clock gating cell

A "gated inverter" for the clock signal is what the NOR gate does. In the absence of an active enable signal (en), the output of the gate (clkb g) will remain a value of '0' irrespective of the input clock signal (logic '0' in the figure). At logic '1' for the en signal, the output clock (clkb g) is the inverse of the input clock. The assertion of the output clkb g occurs when the en signal is active before the clock's rising edge. Using delay buffers, an enhanced reversed clock signal (clkb g del) is produced. When the clock signal happens, the AND gate's two inputs turn into logic '1'. The outcome is the generated pulse. Simultaneously, the clkb g signal will be reduced by the activated clock signal. The resultant pulse will remain high until the clkb g del signal decreases to logic '0' shortly after the buffering delay. The width of the pulse may be altered by adjusting the buffers delay.

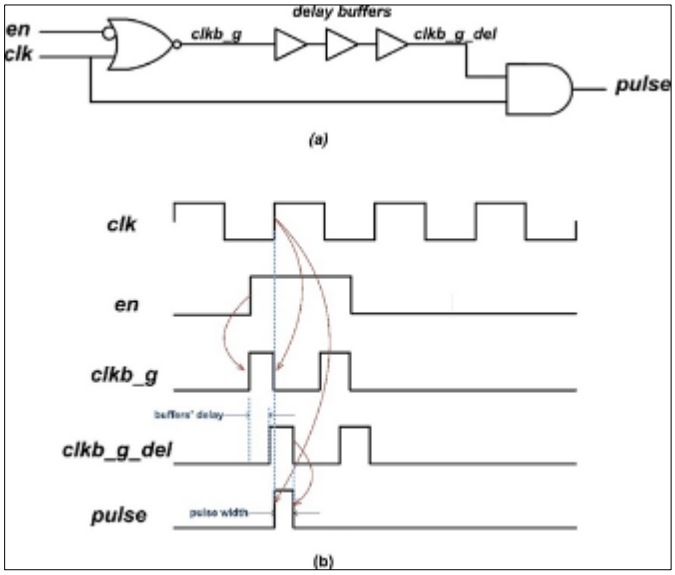

Figure 5 Timing diagram for proposed pulser circuit

There is another design choice available besides the one that was shown above. Two separate input clocks can be physically separated from the pulser's two clock paths (the clock and its delaying variant) due to the data array's regular structure (the identical structure repeated in different registers) and the reduced quantity of pulsers relative to the

quantity of bits (each register having its own pulser). This means that several pulsers can share the delay buffers instead of having them built into each individual pulser, as seen in Fig 6. Space and energy will be further reduced as a result.

This strategy complicates the operations of location, route, and clock tree synthesis (CTS). Instead of transmitting just one clock signal to every pulser, two signals will have to be sent with the requisite skew to alter the pulse width. Also, typical delay buffers need to be carefully picked out and tested in a number of different situations, since any changes or problems with how they work would affect the whole function of the register file, not just one register. The other blocks in the register file will be a lot like the ones that use flip-flops. By applying the wr-enable & wr-address, the write logic creates the pulser enable signals. The read logic, on the other hand, uses the read enable & read address to choose which output data to read. From what was said above, PL-based register files seem to preserve all the positive aspects of latch-based register files while mitigating most of their faults. Consequently, from now on, we will focus on register files that are PL-based, SRAM-based, or flip-flop-based rather than latch-based.

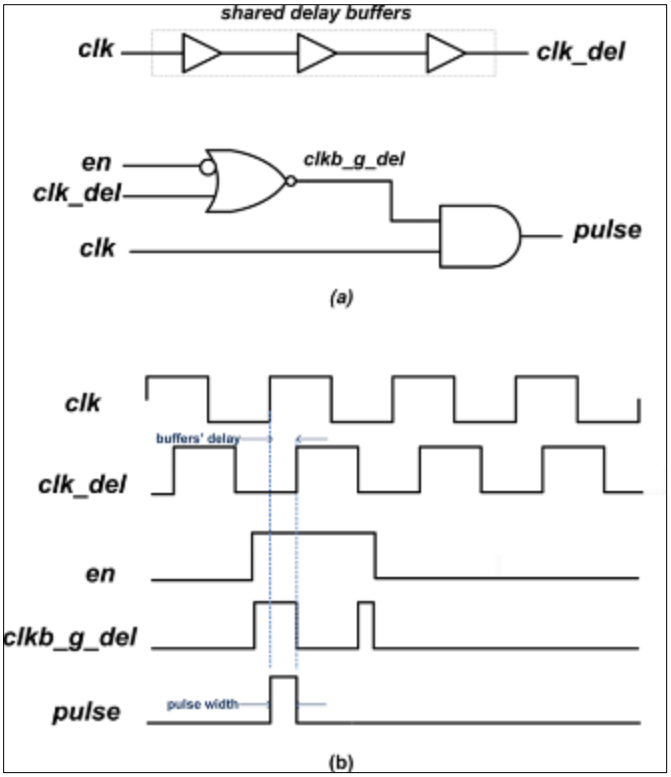

Figure 6 Shared delay buffers are used in modified pulsers. (a) Circuit (b) A timing diagram

2.9. Comparison of 1R1W Register File Implementations

Three distinct implementations for the 1R1W register file have been reviewed: one based on SRAM, one based on flip-flops, and one based on pulsed latches. We have used UMC 28nm technology to implement the three register file options that were described. For the implementation of SRAM, we used the two-port high density register file compiler from Synopsys. Standard threshold voltage transistors are used in the peripheral circuits, while high threshold voltage transistors are used in the bit cells. The standard cells libraries are used to implement the flip-flop and pulsed latch implementations. We employed standard threshold voltage cells for the combinational components while using high-level voltage cells for the storage components (latches and flip-flops) to achieve a suitable alignment with SRAM. Clock gating was employed during the synthesis of the flip-flop design. Following the adoption of these measures and the adjustment of routes, the CTS (clock tree synthesis) was optimized for minimal power use.

Table 1 UMC 28nm 1R1W SRAM bit-cell, flip-flop, and latch area

Bitcell	**SRAM**	**Flip- Flop**	**Latch**
Area(Area (μm^2)	0.4	1.8	1.2

To save space, SRAM bitcells are manufactured with the smallest possible size, in contrast to standard cells which have distinct design limitations (for instance, in order to guarantee alignment during place and route, the majority of cells should have the same height, and the cell length must be a multiple of certain routing grids provided by the design tools). As a result, standard cells are expected to produce storage elements that are substantially bigger than SRAM bitcells. Using the UMC-28nm process, this is shown in Table 1. On the other hand, SRAM secondary circuits take up a lot of space. Smaller SRAM register files may have a significant area overhead. The threshold at which the total size of the SRAM-based register file exceeds that of the traditional cell-based register file may reach up to 1 kbit, contingent upon the word count and bits per word [13]. For this research, we used industry-standard 32-bit word registers and generated many register files with counts ranging from 0 to 1024. Fig 7 displays the post position and route area for four different word counts for the three register file approaches that were explained earlier. Evidenced here is a constant size reduction of over 20% compared to flip-flop equivalents for register files based on pulsed latches. When compared to the register file based on SRAM, flip-flops and pulsed latches have a much higher rate of area growth. Compared to its SRAM counterpart, the PL-based register file is 17% bigger when W = 32. This value grows to almost 2.3X for W = 128 and 4.2X for W = 1024, respectively.

But there's a big difference in the power consumption figures. For the identical register files, the VCD post-place and route power numbers are shown in Fig 8. A clock frequency of 500 MHz is used to write and read data from randomly selected registers over the course of approximately 200 cycles. It has been shown that the pulsed latch register file's power consumption is often 40–50% less than the flip-flop version. Up to 64 words, flip-flop-based register files use less power than SRAM; however, with capacities more than 128 words, pulsed latch-based files may be more power-efficient. The pulsed latch-based register files significantly decrease power consumption at these dimensions. When W=32, the pulsed latch register file consumes about 70% less power than its SRAM equivalent. At W = 64, this value increases to 51%, while at W = 128 it increases to 15%. In supposition, typical cell-based 1R1W register files may have a greater area than SRAM register files, nevertheless, they can use significantly fewer power for minor register files, especially those that use pulsed latches. Because register files are visited often, certain programs may find that a little area overhead is worth the advantage of reduced energy per memory access. Furthermore, the lowest supply voltage at which SRAM can function is much more confined [21]. SRAM bitcells become exceptionally slow at lower voltages because they require high threshold transistors to minimize leakage. Additionally, these transistors will experience substantial variability at lower voltages due to their almost minimum sizes. Hence, typical cell-based register files could be a better match for systems that must function at a lesser supply power. The articulation of typical cell-based register files in hardware descriptive interfaces will also provide design teams with additional control and flexibility. Furthermore, this will facilitate the design's adaptation to alternative technologies.

2.10. Implementation using SRAM

Multiport SRAM systems have found widespread application in numerous architectures. In order to manage numerous reads and stores at once [22] and to allow the synchronized execution of numerous commands in a single cycle [23, 24, 25, 26, 27], multiport register files are frequently required by multi-core processors. A simple method for building multi-port SRAMs is to expand the 8T SRAM bitcell in Fig 1 by adding additional read-write access transistors for each additional port, as seen in Fig 10. A read bitline, two read transistors, and a read word line selector are sacrificed for every additional read port. In order to handle the increased load on the bitlines, larger bitcell crossing coupled inverters will be required when adding extra read or write access transistors. Because more word lines and bitlines need to be wired, the memory capacity may also grow four times with the number of ports increased [11]. By integrating an inverter into each cell, the write bitline may be generated locally, saving wire and avoiding the requirement to send the second write bitline across the memory array [16]. Nevertheless, this extra inverter will cause every bitcell area to increase. The number of physical access ports in a memory cell must be decreased to significantly minimize the space of multiport register files. This will enhance latency and save power in addition to array savings. There are two methods for decreasing the size of the register file: employing numerous banks and time multiplexing. Each approach may be used separately or both can be used simultaneously.

Time multiplexing goes by more than one name: double pumping. With this technique, it is possible to access a port multiple times—sometimes even twice—in a single clock cycle. For example, one write port can be used twice: once at the beginning of the clock cycle and once at the end [28]. Two writing processes will be executed as a result of this. As a result, there will be 50% fewer physical write ports. The read ports are no different; they permit the reading of one register during the first half of the clock cycle and another during the second half. Another choice would be to set the register file's frequency of the clock higher than the CPU core's. Hence, in the period of a single main processor clock cycle, the ports for the register files toggle multiple times. When an array is duplicated, it is split into two or more banks or copies. Each bank has a segment of the read ports (or half of the read ports if the array is repeated twice), and these ports share the write ports. This method is referred to as array duplication. A write action uniformly disseminates

identical data to all banks, but a read operation may occur via any bank, based upon the used read port. Increasing the number of read ports per array by 50% may halve the array size, so using two identical arrays with a reduced amount of read ports may result in a substantial area decrease.

Both methods are capable of being utilized simultaneously. For instance, [11] suggests utilizing two register files, each with two read and one write port (2R1W), to create a register file with 4 read and two write ports (4R2W). Each register file contributes two read ports, resulting in four read ports due to array duplication; nevertheless, the two 2R1W sub-arrays use a single write port collectively. Furthermore, write operations use double pumping, which generates two functional write ports from a singular common write port by accessing it twice inside a single clock cycle, hence facilitating two distinct write operations concurrently in one clock cycle. The finished 4R2W register file was two times smaller than the standard 4R2W file.

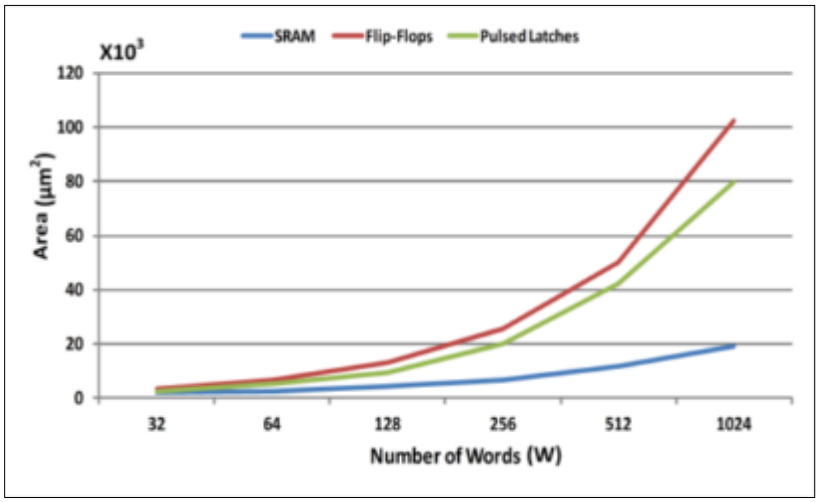

Figure 7 Contrasts the three 32-bit 1R1W register file variations with different word counts

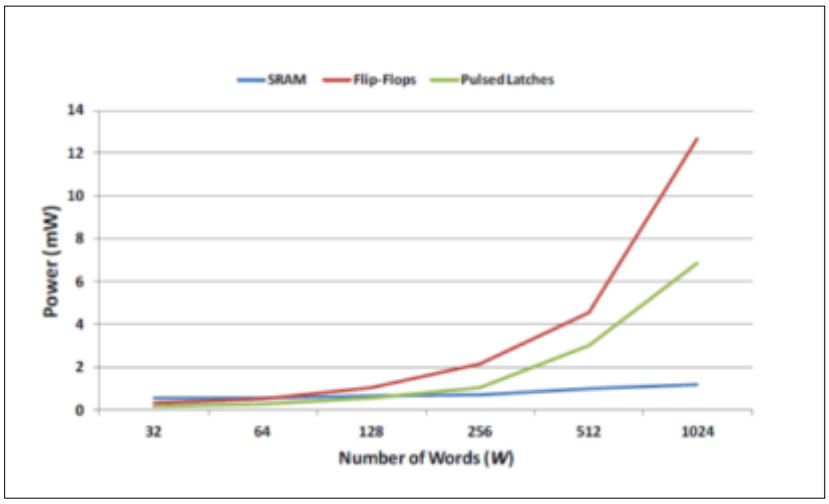

Figure 8 Three 1R1W register files with different word counts and 32 bits each were compared visually

3. Methodology

3.1. Multiport register file design

For superscalar microprocessors, multiport register files are essential [29]. The ability to perform write and read actions simultaneously allows for the implementation of several out-of-order and concurrent multi-threading processes. The register file of the Itanium microprocessor, for example, may simultaneously handle up to 12 read and

10 write operations [30]. A multiport register file with m read ports and j write ports is shown in Fig. 9 as a block diagram. The primary issue with multiport register files is that their space and power consumption both dramatically rise with the number of read or write ports. The several conventional implementations of multiport register files will be covered in the next section.

3.2. RAM Based Implementation

In many different architectures, multiport SRAM systems have been widely employed. Multiport register files allow many instructions to be executed concurrently in a single cycle [23,24,25,26,27], and multiport data caches are typically needed by multi-core processors to manage multiple simultaneous loads and stores [22]. Multi-port SRAMs may be easily constructed by adding more read-write access transistors to the 8T SRAM bitcell shown in Fig 1 for each extra port, as shown in Fig 10. Two read transistors, a read bitline, and a read word line selector can be purchased for each extra read port. Two access transistors, a write bitline, and its complement, and a write word line selector can be purchased for each extra write port. Enhanced bitcell crossing coupled inverters will be required to accommodate the amplified load on bitlines as additional read or write accessing transistors are added. Additionally, the wiring of additional word lines and bitlines may quadratically increase the quantity of memory with the total number of ports [11]. The amount of wire may be reduced by removing the additional write bitline routing in the array of memory and incorporating an inverter in each cell to produce the complementary write bitline locally for each cell [16]. All bitcell areas will, however, significantly rise as a result of this additional inverter.

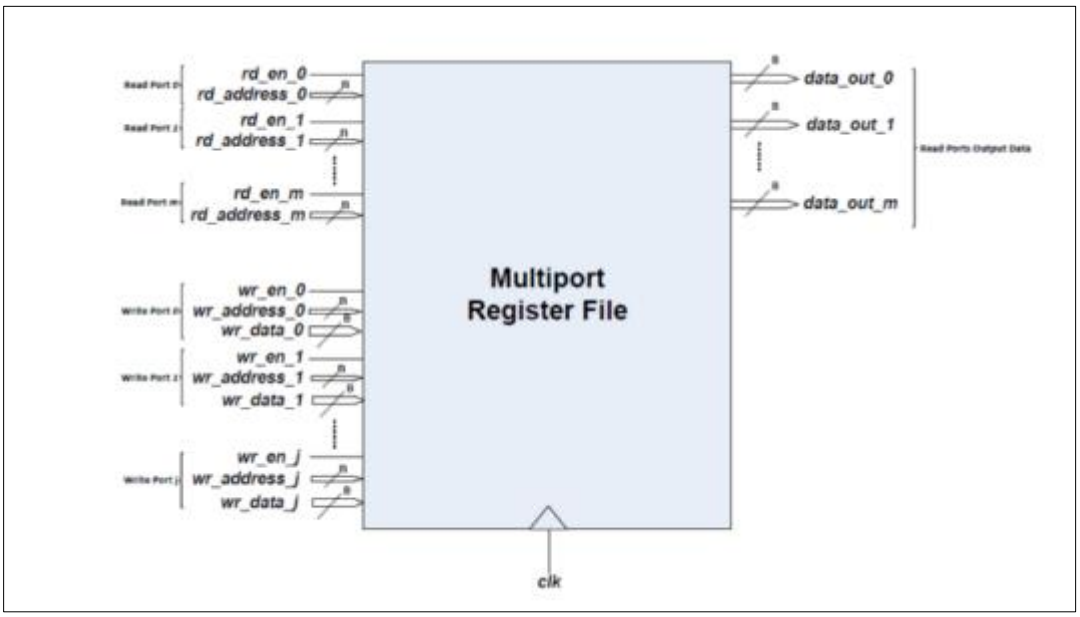

Figure 9 m read and j write multiport register file block diagram

To significantly decrease multiport register file size, lower the number of physical access ports in a memory cell. Power and latency savings will accompany array savings. Multiple banks and temporal multiplexing reduce register file size. Both techniques can be used simultaneously or separately. Double pumping is time multiplexing. A port can be accessed twice in a clock cycle using this method. A single writing port can be accessed twice in a clock cycle [28]. Two effective write operations result. This halves the physical write ports. The first half of the clock cycle allows the read ports to read one register, while the second half allows them to read the other register. Another approach is to operate the register file at a higher clock frequency than the CPU core. Thus, the register file ports toggle many times every main processor clock cycle.

The technique of multiple banks, also referred to as array duplication, involves the division of the memory array into numerous copies, or "banks." Each copy is assigned a portion of the read ports (or fifty percent of the read ports in the scenario of a double-duplicated array), while each copy is permitted to share the write ports. . While reading can happen from any bank (based on the read port selected), writing always writes the same data to all banks. Since adding more read ports to an array might triple or quadruple its size, having two similar arrays with half as many read ports can significantly minimize the amount of space needed.

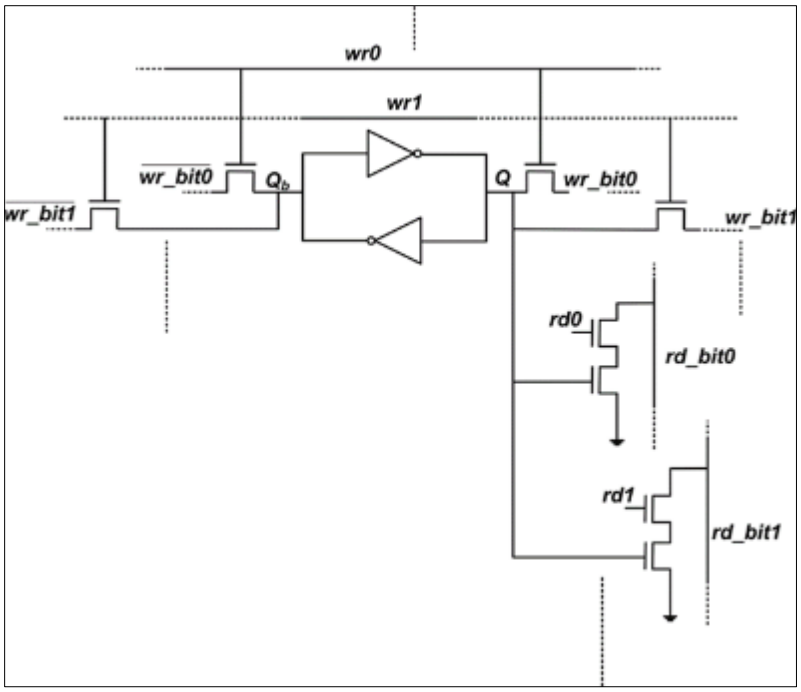

Figure 10 m read, j write multiport register file block diagram

The two techniques may be combined. So, for example, to construct a register file having four read and write ports (4R2W), it is recommended in [11] to use two 2R1W register files. Two 2R1W sub-arrays share a single write port in this array duplication situation, and the two read ports out of each register file are combined to create four read ports. Furthermore, two write operations can be performed in a single clock cycle thanks to double pumping, which involves accessing the single common write port twice. This technique effectively creates two write ports. When compared to a standard 4R2W register file, the resultant file was twice as small in size.

3.3. Application Based on Existing Cell Standards

It is also possible to add more read/write ports to standard cell-based register files, such as pulsed latches and flip-flops. To accommodate the greater cell loads caused by the additional ports, the data array structure—the primary distinction between register files based on pulsed latches and those based on flip-flops—may be substituted with higher driving cells of the same kind. Read and write logic will undergo the most significant alterations. Adding more read ports is as simple as installing a second multiplexer in parallel with the one in Fig 2. When dealing with a high number of bits per register and/or registers, a considerable area overhead is produced by connecting their outputs to the multiplexers' inputs on the read ports. This is particularly true when the number of read ports is considerable. We can easily increase the number of read ports by installing an additional multiplexer parallel to the one in Fig. 2. The interconnection of register outputs to the inputs of the read port multiplexers accumulates significant spatial overhead when dealing with a large number of registers or bits per register. When there are a lot of read ports, this becomes even more apparent.

Adding read or write ports will degrade performance, increase power consumption, and increase area, much like SRAM. Another method to decrease the number of physical ports is array duplication and double pumping, which is comparable to SRAM. Both methods will still have substantial area and power overheads, though. Furthermore, there will be a cap on the amount of ports that can be obtained by duplicating or pumping. By utilizing the two segments of the clock cycle for double pumping, two functional write ports can be achieved. But it might be trickier to have more than two write ports without incurring a lot of extra expense. Similarly, if you need more read ports, array duplication could be a solution. But there's a huge space penalty for making copies of the data array. Additionally, the power consumption increases with duplicate arrays due to dynamic power (from writing the exact same data in multiple data arrays simultaneously) and leakage power (from additional bit storage components).

3.4. Virtual Ports and a Pulsed Latch-Based Register File

When implementing small to medium register files, pulsed latch based files can offer some benefits, as mentioned in section 2.6. Along with reduced latency and smaller footprint compared to flip-flop based register files, they can significantly reduce power consumption. But, like with previous register file designs, incorporating read/write ports

can lead to performance loss, power consumption spikes, and substantial space overhead. Pulser circuits may be used to provide access to the same register several times in a single clock cycle since pulsed latches depend on them to create short pulses via clock signals. Furthermore, additional pulsers can be organized into groups, which we refer to as Pulser Groups (PGs), to carry out specific tasks required for read or write transactions. Fig 11 depicts the result: a register file with several virtual ports produced from a relatively small number of physical ports via the employment of control logic blocks to perform both read and write operations. Several pulser groups are used in these control logic blocks to generate control signals. Additionally, they store intermediate data, including internal read and write addresses and data. What follows are two parts that go into this issue at length.

3.4.1. Implementing Virtual Read Ports

The read logic that generates the data output for every reading port is often achieved by parallel W-to-1 multiplexers, as described in section 2.6. These multiplexers' outputs are the required output data, and the read address acts as a selection. These multiplexers must be replicated for each extra read port in order to utilize a traditional register file with multiple physical read ports. Nevertheless, our proposed architecture will employ a single multiplexer to produce the necessary data for each of the virtual terminals on multiple occasions within a single clock cycle. This can only be achieved by logic-based selection of an input read address for an enabled read port. In addition, additional logic is required to retain the routed read data for each port. Detailed block diagrams for our proposed read logic are illustrated in Fig 12. It has five principal components, including the primary data multiplexer, identical to that found in the 1R1W register file.

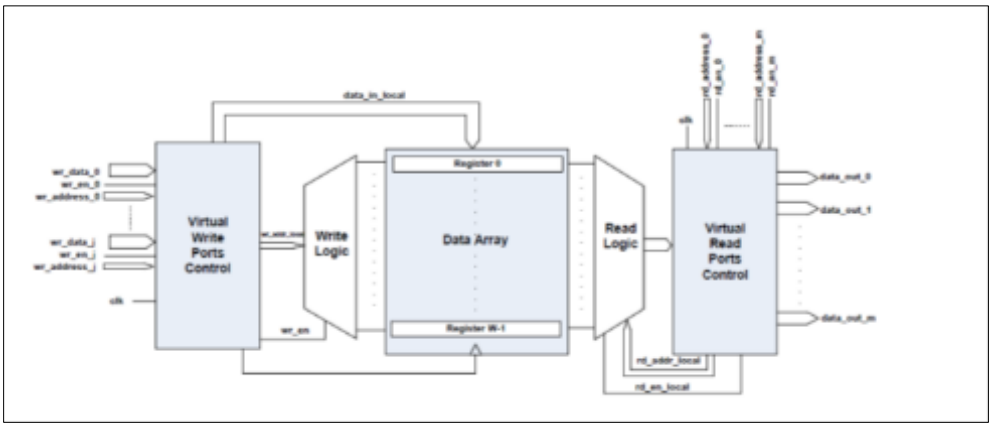

Figure 11 Virtual port-based pulsed latch register file structure

3.5. Clock Generator Internal

Producing clock signals having phases different from the principal input clock signal (clk) is the duty of this block. When no read operation is needed, as indicated by the lack of any read enable signals (rd en's), it is made up of a clock-gated cell that controls the clk signal. If no less than one of the rd en is active, the clk signal is sent to clk g. The read logic will use this signal as its main clock signal. Utilizing delay buffers, the clk g is used to produce several clock signals (clk g del) with distinct phases. To regulate the read operation for each read port, the other four blocks employ these clk g del signals.

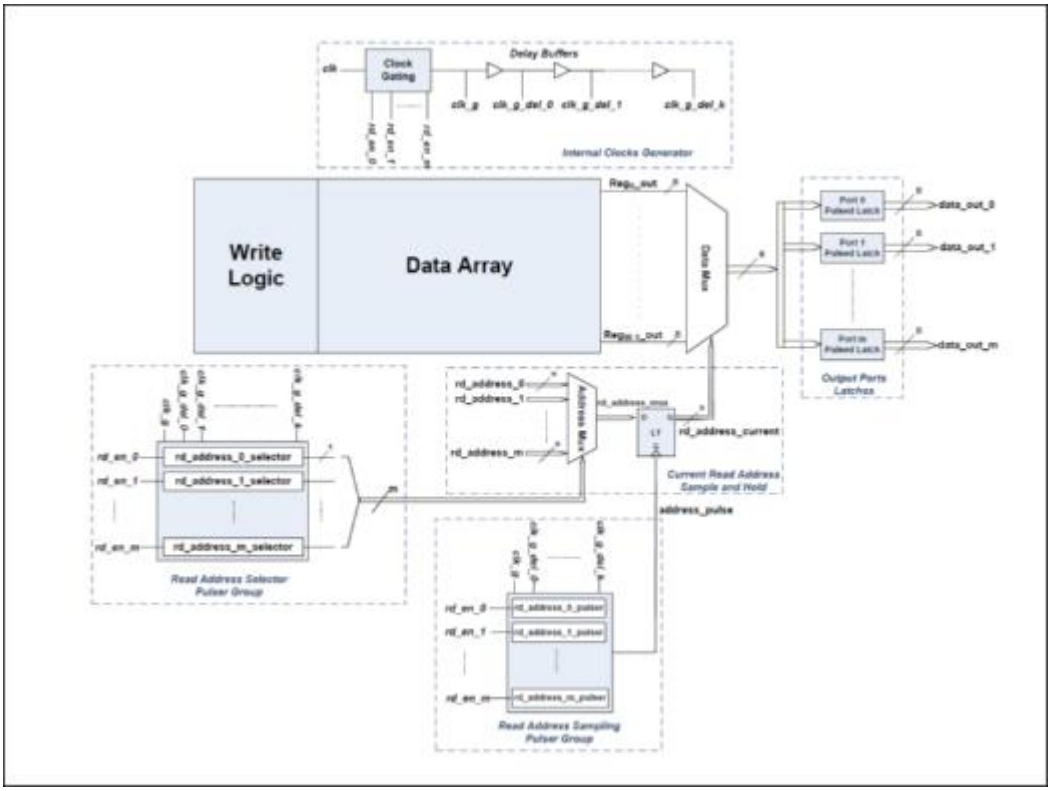

Figure 12 The read logic structure that is being proposed to provide multiple virtual read interfaces

3.6. Pulser Group for Address Selector Reading

The Address Mux selects the input read addresses to read next based on the control signals generated by this block. For every read port, it has m sub-blocks. Each of the building blocks is a pulser circuit with a layout similar to the one in Fig 6 (a). Two clock signals come from the internal clock generator and one input enabling signal is the rd en of each pulser's associated port. One of the Address Mux selectors produces a pulse as its output.

3.7. Group for Reading Address Sampling

This block allows one to create the address pulse signal. This signal triggers the latches for reading addresses found in the Current Read Address Sample and Hold block. The functional viewpoint holds that a large "pulser" circuit may generate a limited number of pulses in one clock cycle. Though it contains two produced clock signals instead of an input enable signal, it has m pulser circuits, same like the read address selector pulser group. For the read address latches, the enable signal is produced by combining the outputs of several m pulsers.

3.7.1. Sample of Current Read Address and Hold

This block must first choose an input read address and maintain it for a sufficient duration to enable the data multiplexer to decide and send the necessary register data to the latches on the ports for output. It is composed of two sub-blocks: the Address Mux and the Address Latch. The Address Mux, a collection of multiplexers, selects one of the input read addresses based on the choice signals supplied by the read address selector pulser group. The read addresses on ports 0, 1, and so on are given higher priority by arranging this collection of multiplexers in a specific order. Additionally, if no selectors are set, the Address Mux will automatically display the read address of port 0. Fig. 13 shows how this Address Mux is laid up. The address of port 0 is selected if the read port 0 selection is activated. In that case, the read port selector is checked by the multiplexer, and so on. When none of the selectors are in use, port 0 is chosen automatically. Port 0 is used as the default address to reduce the setup time for read-enable signals since it is the primary virtual port in the clock cycle to be checked if enabled. The address latch is responsible for storing the address of the resistor that is currently being read. It is necessary to hold the current read address constant long enough for the Data Mux to transfer the data from the chosen registers to their output and then for one of the latches on that output port to save that data. A signal known as an address pulse activates this latch. For the current read address, the Address Latch and the Read Address Sampling Pulser Group constitute a pulsed latch register.

3.8. Ports for the Output Latches

During a single clock cycle only one data multiplexer is employed to read out data from various read ports, for its output to be one of the register file's actual read port outputs, it must be placed at the right time throughout the cycle. Additionally, each read output must remain stable until a new read operation is performed on the same port. This is achieved by using the Output Ports Latches block. The block contains a limited quantity of periodic latches, each with an identical quantity of read ports. A clock signal produced by the clock's internal generator resumes each pulsed latch once it has been activated by a read-enable signal. The choice of the triggered clock will help to guarantee that the data mux output includes suitable data for storage.

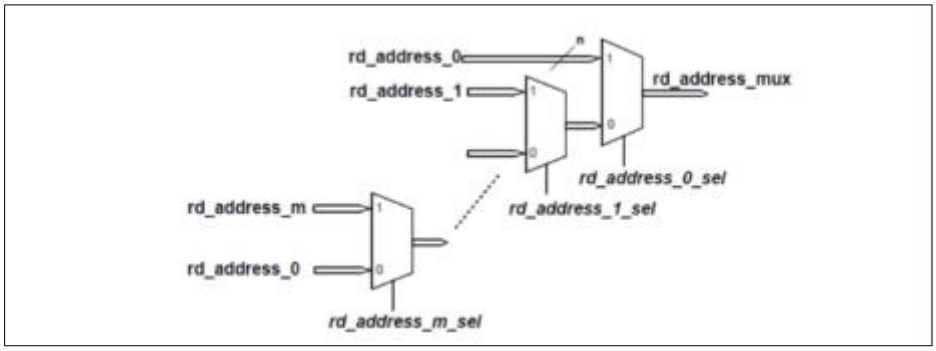

Figure 13 Address Mux structure emphasizing read ports in sequence

3.9. Implementing Virtual Write Ports

As explained in section 2.4, the write logic block is accountable for activating the pulser of a specific register inside the data array of a register file that employs a pulsed latch. Usually, a decoder can do this by activating one of its outputs in response to the wr address after being triggered by the wr en signal. For a single decoder to support numerous write operations, control logic is needed to activate each write address sequentially before sending it to the decoder circuit. The same control logic should also decide which inputs to write and send them to the data array simultaneously with its present write address and enable signal since all of the data array's inputs are identical. The Port Selector is the name we give to this control logic. One multiplexer manages the write address, another handles the write enable, and the third manages the write data. The multiplexer select signal is generated by the Write Port Selector block, which is responsible for controlling these multiplexers. The block derives its signals from the internal clock generator.

Fig 14 displays the suggested write logic design. To create the data array's write clock signal, you'll need an additional control block in addition to the Port Selector. If you want to execute numerous write operations in a single clock cycle, you'll need to trigger the clock signal more than once. This is because the data array's register pulsers use the triggering clock signal to generate the enable pulses for the latches. This is made possible by the Data Array Clock Generator, which, whenever a write operation is required, activates the data array's clock signal (clkw data array). Based on the current write address, the write decoder ought to have already enabled the required register prior to the triggering event. Additionally, the data array should already have the current write data.

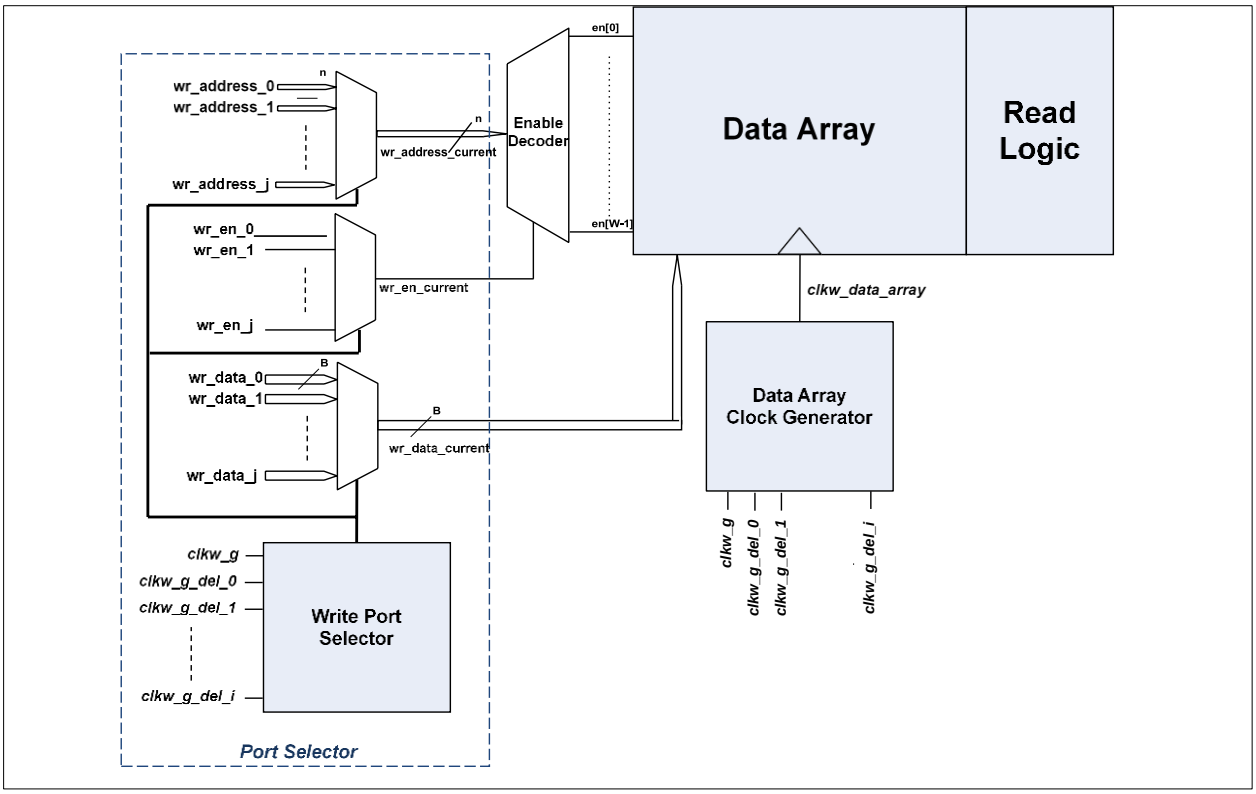

Figure 14 Multiple virtual write ports in the proposed write logic structure

4. Results

4.1. Register Circuit Designs Files

To evaluate the different methods for implementing multiport register files, we used all of the available implementations to create 32-word register files, each with 32 bits, and a small number of multiport configurations (2 or 4 read ports with 1 or 2 write ports, for example). Alongside the 1R1W configuration, four additional multiport configurations were evaluated, including a 2R1W register file featuring 2 read as well as 1 write port, a 2R2W register file comprising 2 read and 2 write ports, a 4R1W register file with 4 read and 2 write ports, and a 4R2W register file containing 4 read and 1 write port. Nonetheless, comparable design concepts may be implemented for different quantities of write and read ports.

An instance of 1R1W was created for the SRAM executions utilizing the UMC 28nm register file compilers. Using CACTI [31], a software for modelling dynamic and leaky power, accessing time, and area of cache and other memory types, we first scaled the 1R1W values in order to get the required area metrics for various actual multiport setups. The 1R1W power values were adjusted utilizing the CACTI metrics following the application of the power analysis tool to derive approximate energy for each write and read port of 1R1W. After that, the average power usage for a specific activity in the register files was calculated by adding the scaled numbers according to the action retrieved from VCD files. Additionally, we used array recurrence and double pumping on the 1R1W memory array to compare our proposed methods to current solutions for multiport SRAM register files. Using a shared write port, this array was duplicated twice to produce the 2R1W register file. To obtain the 4R1W register files, it was necessary to duplicate the memory array four times using the same write port. By incorporating extra circuitry to execute both write operations inside the same clock cycle, a technique known as "double pumping" was employed to achieve two write ports. Both write operations occur throughout the clock cycle, however, the first one happens in the first half and the second in the second. The 2R2W and 4R2W register files were obtained using this dual pumping in conjunction with the array replication that was formerly covered.

In order to construct various designs in RTL utilizing the UMC 28nm libraries, standard cell-based register files were examined. The clock tree synthesis (CTS) operation was completed along with the synthesis, placement, and routing of all the designs. The design was augmented from the 1R1W architecture seen in Fig 2 for traditional cell-based register

files that included actual ports. Each new write port required a supplementary write decoder, together with hardware to manage the activated signals and data inputs to a data array. The addition of read ports was achieved by adding additional multiplexers in parallel. A flip-flop data array and a pulsed latch data array were both included in the two standard cell register files that were applied to each multiport arrangement.

Utilizing virtual ports and pulsed latches, the recommended register file was implemented. Elaborated in Fig 15 is the read logic that provides two read virtual ports. Two gated clock variants, one with a delay of 0 and the other with a delay of 1, are required to support two read interfaces. Additionally, you may use the rd en 1 and clk g del 0 signals to directly specify which read address to be read at a specific time, avoiding the Read Address Selector Pulser Group. If both signals are set to '1', port 0 will be utilized by default, however this may be altered. Every time the Read Address Sampling Pulser Group generates a pulse signal, the rd address current latches samples the given address. The required address pulse signal is generated by ORing the outputs of the two pulser circuits that make up this pulser group. Both pulsers are activated by the read-enable signals originating from the two separate read ports. Before choosing the event-triggering clock for each pulser, the necessary read address is picked and directed to the latch inputs, as seen in Fig. 15 of the timing diagram. Data from the output was stored in two pulsed latches. Finally, each port's read enable signals were selected to turn it on, and the triggering clocks were set up to only activate the pulsed latches once the data out signal was successfully routed through the required read data. When the clk g del 1 clock signal rises, the pulsed latch on port 0 will activate, and when the clk g del 0 clock signal falls, the pulsed latch on port 1 will activate. By modifying the read logic as illustrated in Fig 16, four read ports can be obtained. The generation of an extra delayed clock signal is accomplished by incorporating a third delay buffer. A 4-bit signal is generated through the combination of the rd addr 0 pulse signal obtained from the Read Address Sampling Pulser Group with the three control signals from the Read Address Selector Pulser Group. One of the four read addresses is selected by the Address Mux's selector using this signal. In accordance with the approach specified in Section 3.7, the ports are assessed in a prioritized sequence. The rd address 0 is selected when the rd addr 0 pulse signal is active. Then, the first available RD address is used. If it isn't, the second address will be chosen if the second selector for RD is activated. Finally, rd address 3 is chosen if there is just one active signal, that is, rd addr 3 sel. In the absence of any of the four signals, rd address 0 is chosen automatically.

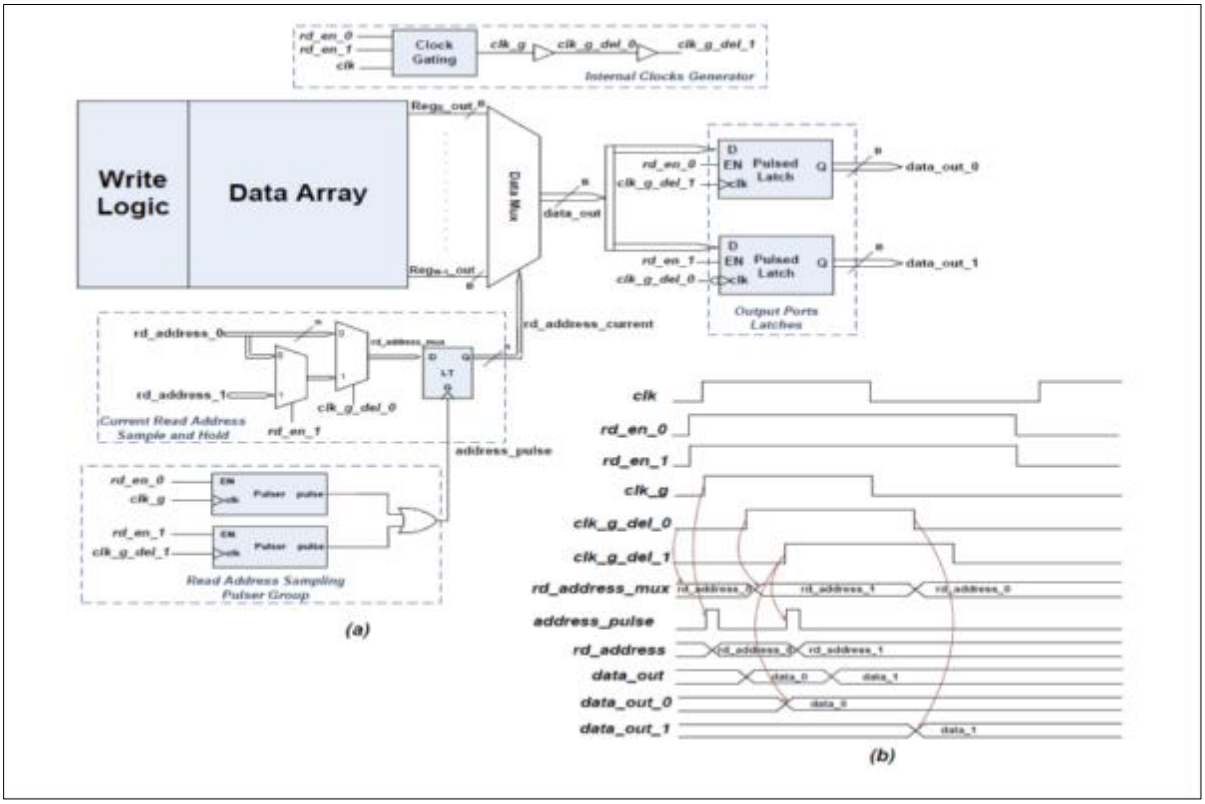

Figure 15 is the suggested architecture for the read logic of register files that have two virtual read ports

The address pulse signal is generated by ORing the four pulser circuits that make up the Read Address Sampling Pulser Group. One of the read-enable signals activates each of the four pulsers, and two clock signals having different phases are selected to supply the necessary pulse signal. Two clock signals are used; one originates the pulse and the other

stops it. The two clock signals are selected in such a way that the necessary read address is pre-picked, sent to the address latches' input, and maintained stable during the pulse. Each port's output data is stored using four pulsed latches, two of which are activated at the rising edge of the clock signal and the other two at the falling edge. After being triggered by the corresponding clock signals, the read enables signals activate each port. This ensures that the data out is prepared with the required read data, just like in the two-port configuration. Fig 16(b) shows how the read logic functions overall if four read operations are carried out immediately. The purpose of the read logic is to ensure that all read data is stable at the output ports before the next clock cycle. This should allow adequate time for preparation for the subsequent step, which reads the register file's output.

Two write ports' logic designs are shown in Fig 17. To do two write operations in parallel, you need one delayed clock signal and one gated clock signal. The port selection process requires three write enable, write address, and write data 2-to-1 multiplexers. You can think of the gated clock signal clkw g as the selector for the three multiplexers. This means that every port is chosen for half a clock cycle. The enable decoder would turn on a data array register if the chosen port is enabled. The clock signal for the data array, referred to as clkw data array, is generated using the AND-OR configuration. When the write port is enabled, or the write current is '1', a clock signal is produced to turn on the chosen register's pulsed latch. Fig 17(b) of the timing diagram illustrates how to ensure a correct write operation by activating the clkw data array sufficiently long after the write address is selected and data is moved to it. Two intervals are indicated in each half of the clkw g signal in the timing diagram.

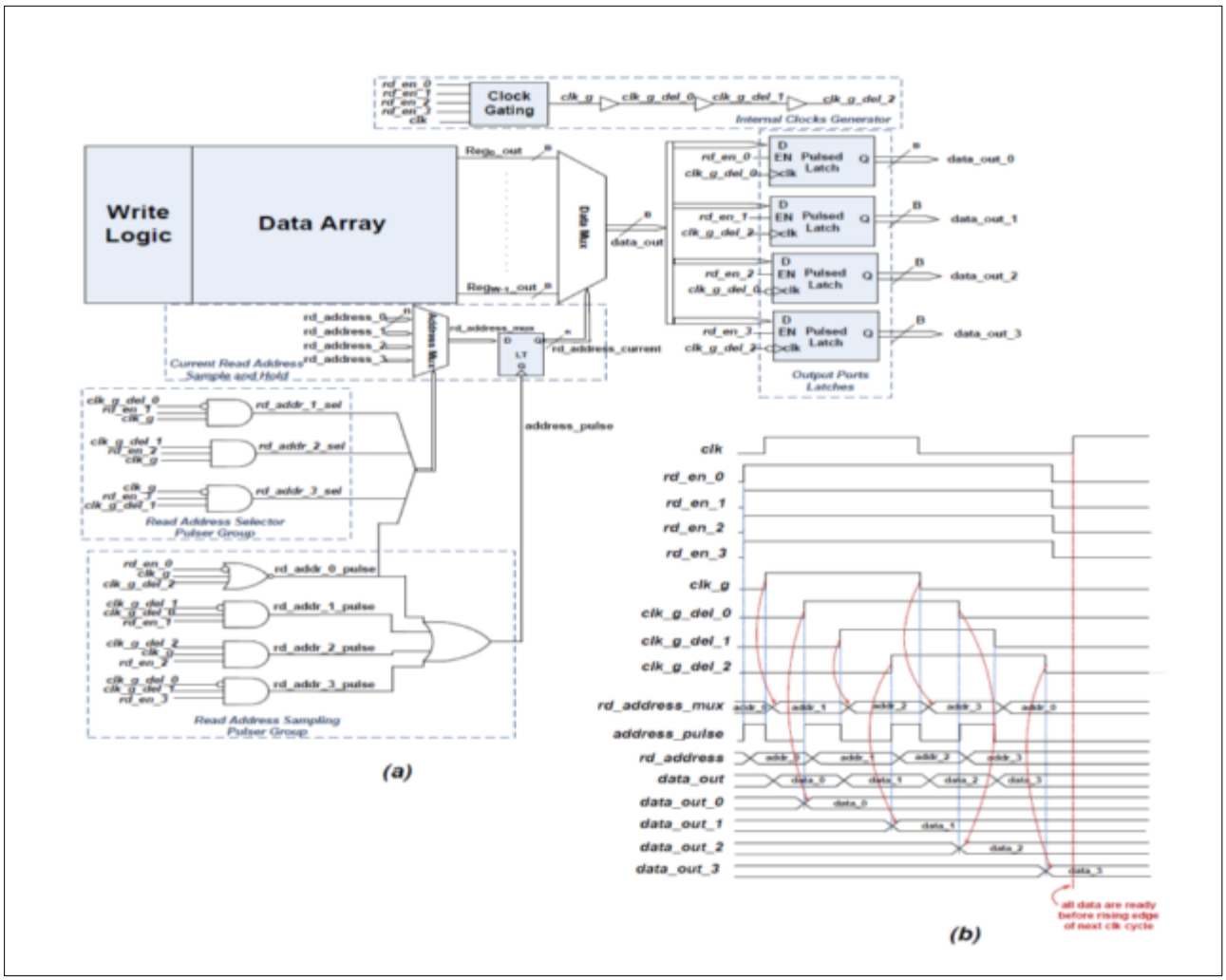

Figure 16 Our proposed register file reading logic with four virtual read ports

A register is enabled by the enable decoder when the port selector selects the writing port. New write data is captured and stored by the data array at the second interval. Every design was low-area and power. A Synopsys Design Compiler synthesized the designs. We used Synopsys IC Compiler for location, route, and clock tree synthesis. Synopsys VCS

simulated all designs on different testbenches at 500MHz. Post-place and route simulation VCD files were utilized for power analysis in Synopsys Prime Time PX.

4.2. Register Files Area

Several register file executions with different port counts are shown in Fig 18, along with their respective area numbers. Certain area advantages are provided by the 1R1W SRAM register file, The area, however, begins to grow rapidly upon the integration of additional read-or write-oriented ports. The size grows by almost 30 to 40 % for each extra write port and almost two times for each additional read port. The significant expansion results from the wiring of the read and write bitlines [11]; the increased size of SRAM bitcells guarantees proper read and write capability with the enhanced bitline capacitance. When compared to a single memory array, the register file space that comes from using array duplication is smaller. Additionally, compared to having a second physical write port, employing dual pumping for write operations only results in a small area overhead. As additional read/write ports are added, the fundamental cell-based register file expands in size, much like static random-access memory (SRAM). However, compared to SRAM, the area expansion rate is far slower. The register file based on flip-flops is consistently smaller than its SRAM equivalent, with the exception of the 1R1W register file.

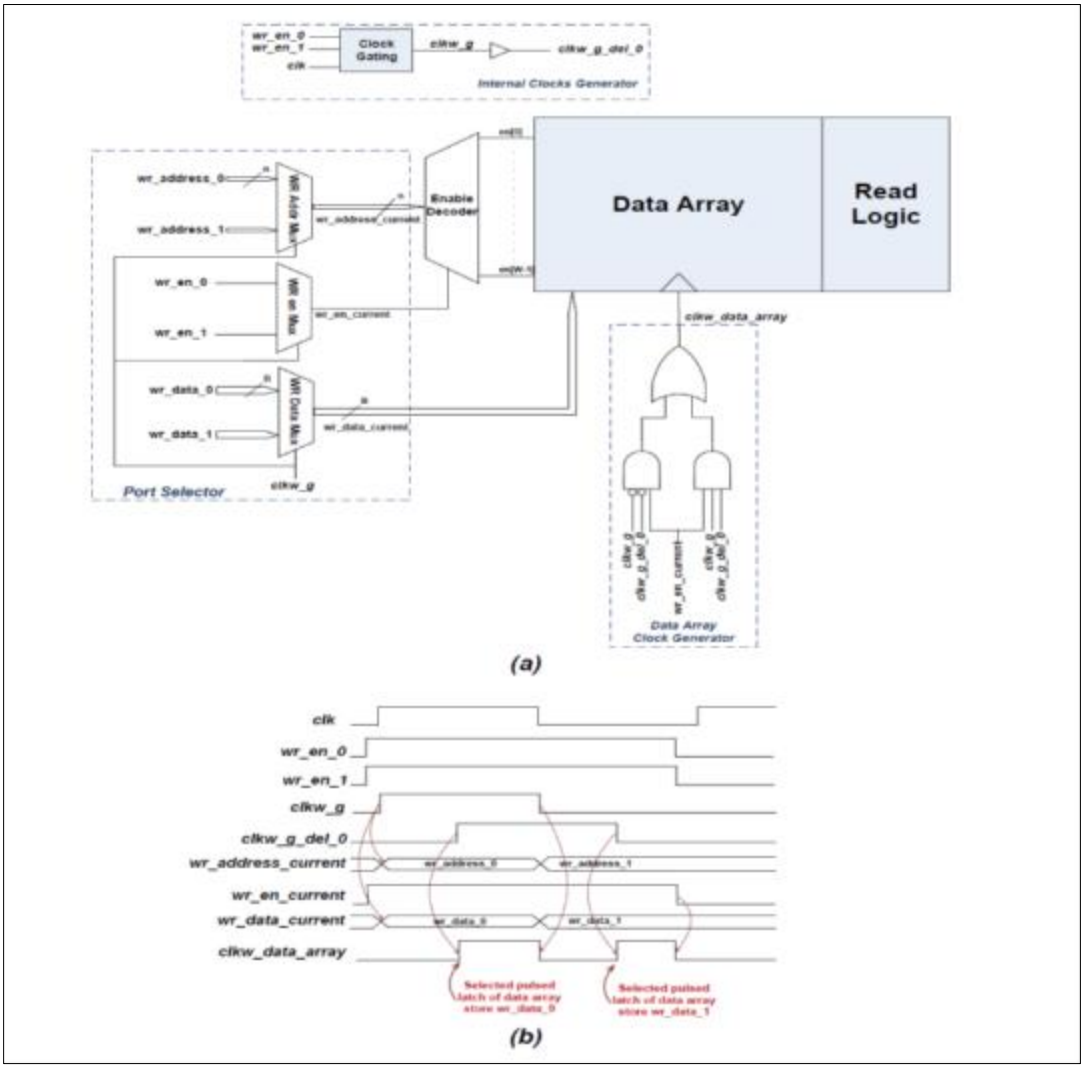

Figure 17 Our suggested register file write logical framework with two virtual write ports

However, the flip-flop-based register file exhibits specific area gains for read ports numbering more than two when array replication is used. Moreover, when array replication and dual pumping are used on SRAM, there is a noticeable area advantage in all variants of the pulsed latch register file with physical ports (with the exception of the 1R1W). However, a considerable amount of space is still required for each extra port (4R2W register files are more than 2.4 times larger than 1R1W files).

Here comes one of the benefits of our suggested approach. Fig 18 shows a minor area overhead introduced for each additional read or write port. A 4R2W pulsed latch register file, for example, has an area that is just 20% larger than the 1R1W register file due to our virtual port design. The use of pulsed latches or flip-flops contributes around 15% to the area overhead required to achieve an equivalent number of physical ports, but using SRAM results in just 3% space overhead. The same proposed technique may be used to create additional read or write ports. Based on the operating clock frequency and the relevant technology, the maximum number of virtual ports that may be used with a single read or write port is established. It is possible to add a second port and apply the same read/write logic on both if the required number of ports is more than the supreme number that can be used with a single physical port. This will lead to a significant decrease in floor area relative to the other implementation alternatives.

Furthermore, because of the relatively low area overhead of the 4R2W register file in comparison to the 1R1W, the 4R2W architecture may be used generally in all designs. Depending on the operating application, the processing unit will have the liberty to use as many ports as necessary at run time. Having several read and write ports could help in the processing time of programs with several independent processes that can be handled in parallel.

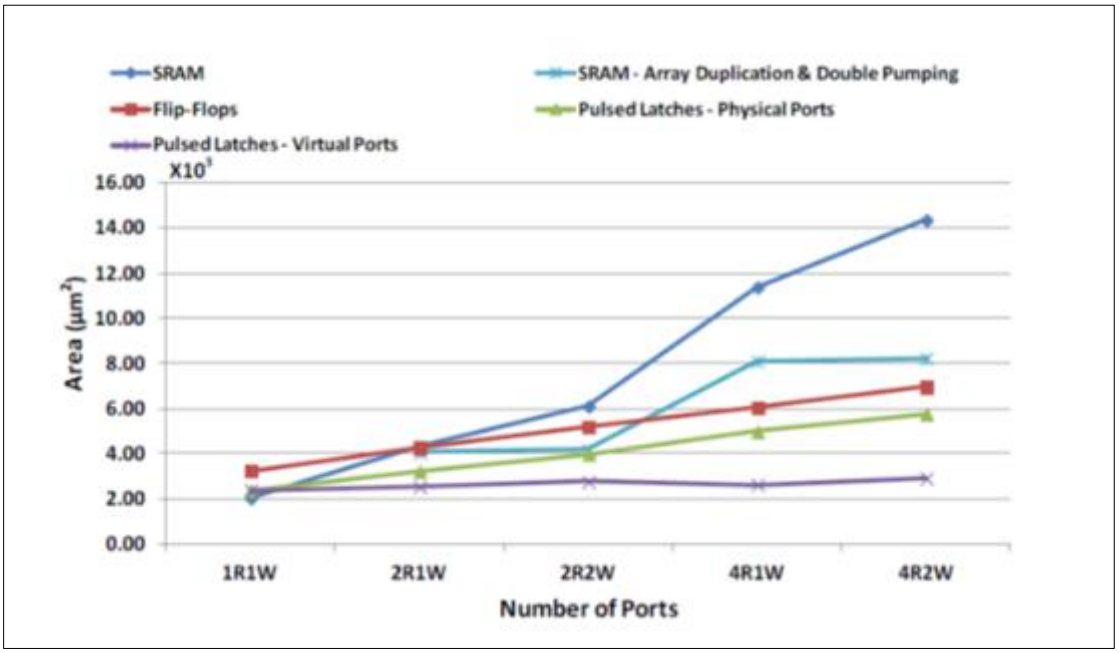

Figure 18 Comparative analysis of the area of five variants of the 32-bit x by 32-word register files with varying read and write ports

4.3. Register Files Power Consumption

We estimated the average energy consumption of register file structure by performing post-layout power evaluations using VCD activity gathered from examine benches made for each register file configuration, in which data is read from and written to random addresses. This was executed at a clock frequency of 500 MHz for over 200 cycles. Fig 19 illustrates the power usage measured for various register file executions with differing port counts. The power usage of SRAM consistently exceeds that of conventional cell-based systems, as anticipated for a 1-Kbit register file size. The energy consumption of the 4R2W SRAM register file is more than 2.5 times that of register files based on flip-flops or pulsed latches. Using array replicating and dual pumping results in a small rise in the SRAM register file's power usage.

In comparing typical cell-based solutions with the two pulsed latch applications, the flip-flop-based register file consistently uses more power. The power consumption of shared pulser flip-flops is anticipated to be greater than that of pulsed latches. The increased input clock demand of flip-flops correspondingly amplifies power consumption in the clock network. In comparison to our proposed virtual port approach, the pulsed latch execution with a separate physical port for read/write operations uses less power than the other one. The power overhead of the mentioned method is just 7% more than its physical port counterpart on average, regardless of the number of ports. With a difference of less than 10%, the pulsed latch register file with virtual ports uses 9% less power on average than flip-flop-based alternatives.

Applications that need fewer ports may benefit from the universal use of the 4R2W pulsed latch register file with a virtual port. However, the additional power usage is unlikely to be substantial. The proposed 4R2W register file's power consumption during the execution of similar test benches used for the power analysis of different port counts is shown in Fig. 18. The power overhead is around 2%, except for the 1R1W configuration, as illustrated. Even though there is an 87% power overhead compared to a specialized 1R1W pulsed latch register file, the 4R2W register file operating as a 1R1W will use more than 10% less power than a flip-flop-based 1R1W register file.

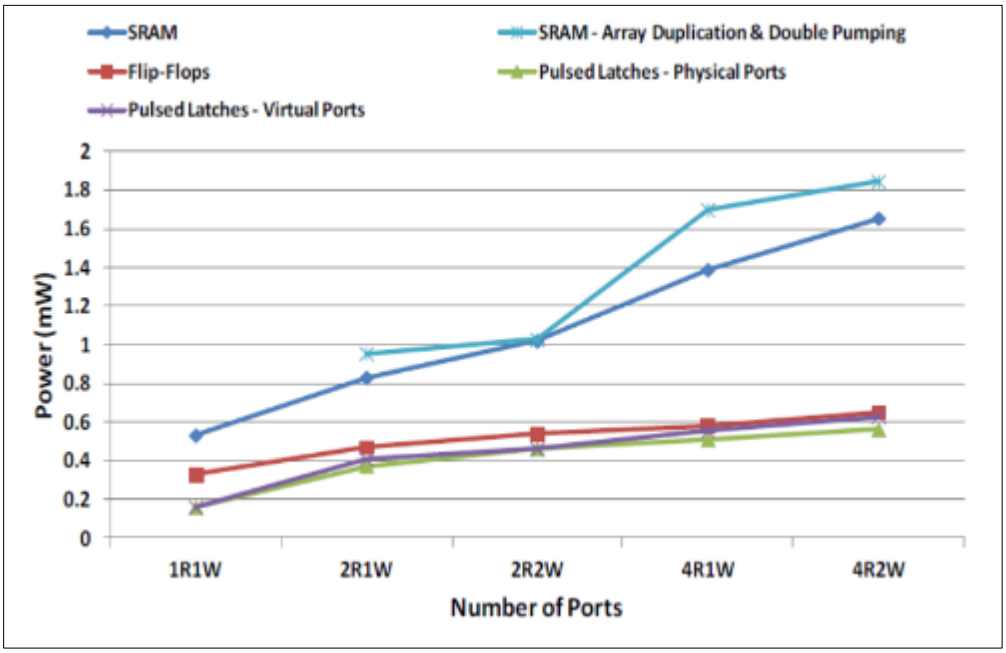

Figure 19 The 5 register file implementations with varying port numbers and their respective power consumption

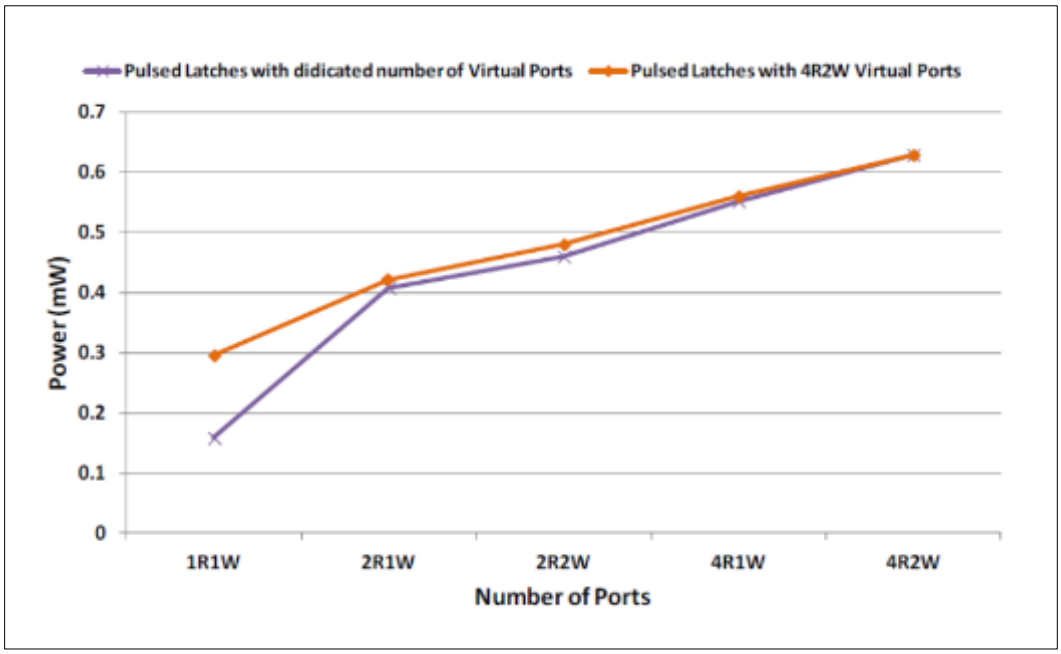

Figure 20 Pulsed latch-based register file power consumption with 4R2W virtual ports compared to lower port count testing benches

List of Abbreviations

Acronym	Abbreviation
SRAM	Static Random-Access Memory
FPGA	Field Programmable Gate Arrays
STT-RAM	Spin-Transfer Torque Random Access Memory
FF-RF	Flip Flop-Based Register File
CGC	Clock Gating Cycle
PL	Pulse Latch
CTS	Clock Tree Synthesis
Wr	Write
Rd	Read

5. Conclusion

We examined various methods of execution of register files. The conventional methods of implementing 1R1W register files utilizing SRAM, flip-flops, and latches were compared in order to offer a standard cell solution that is efficient for small register files. The final decision was to use a pulsed latch architecture. In addition, we discussed several approaches to multiport register file layouts that increase area efficiency.

We covered the specifics of adding a virtual read/write port with a reduced number of physical ports and offered a new pulsed latch implementation for multiport register files. Various multiport register file implementations were showcased and evaluated, each with a unique number of ports. The proposed approach with virtual ports led to considerable space savings compared to previous systems employing SRAMs, flip-flops, or pulsed latches with physical ports. Furthermore, the recommended solutions are 7% more power efficient compared to the pulsed latch-based register file that uses physical ports, demonstrating their exceptional power efficiency. Its power consumption, but, is consistently lower than that of its flip-flop or SRAM equivalents.

In order to test how well the suggested virtual port approach works with 4R2W register files, we looked at them as a generic implementation of register files that can be adjusted at runtime to use the number of ports needed by the program. The results obtained with fewer ports for the register file show a minimal energy overhead when compared with other register files having specific read and write ports. Additionally, the area overhead is just around 20% when associated with a dedicated 1R1W pulsed latch register file.

Compliance with ethical standards

Disclosure of conflict of interest

No conflicts of interest have been disclosed by any of the authors.

References

[1] 1 V. Zyuban and **P.** Kogge. The energy complexity of register files. In Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No.98TH8379), pages 305-310, Aug 1998.

[2] F. Oboril and M. B. Tahoori. Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012),* pages 1-12, June 2012.

[3] **H.** Kaeslin. Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fab- rication. Cambridge University Press, Cambridge, 007 2007.

[4] P. Meinerzhagen, S. M. Y. Sherazi, A. Burg, and J. N. Rodrigues. Benchmarking of standard-cell based memories in the Sub-Yr domain in 65-nm CMOS technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* 1(2):173-182, June 2011.

[5] S. Bernard, M. Belleville, J.-D. Legat, A. Valentian, and D. Bol. Ultra-wide voltage range pulse-triggered flip-flops and register file with tunable energy-delay target in 28 nm UTBB-FDSOI. Microelectronics Journal, 57:76 - 86, 2016.

[6] D. Bol, J. D. Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and

[7] J. D. Legat. Sleepwalker: A 25-MHz 0.4-V sub-mm^2 *7-μW* /MHz microcontroller in 65- nm LP /GP CMOS for low-carbon wireless sensor nodes. *IEEE Journal of Solid-State Circuits,* 48(1):20-32, Jan 2013.

[8] F. Botman, J. de Vos, S. Bernard, F. Stas, J. D. Legat, and D. Bol. Bellevue: A 50MHz variable-width SIMD 32bit microcontroller at 0.37v for processing-intensive wireless sensor nodes. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS),* pages 1207-1210, June 2014.

[9] J. L. Cruz, A. Gonzalez, M. Valero, and N. P. Topham. Multiple-banked register file ar- chitectures. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201),* pages 316-325, June 2000.

[10] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi. Reducing the complexity of the register file in dynamic superscalar processors. In *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34,* pages 237-248, Dec 2001.

[11] A. Aggarwal and M. Franklin. Energy efficient asymmetrically ported register files. In *Proceedings 21st International Conference on Computer Design,* pages 2-7, Oct 2003.

[12] G. S. Ditlow, R. K. Montoye, S. N. Storino, S. M. Dance, S. Ehrenreich, B. M. Fleischer, T. W. Fox, K. M. Holmes, J. Mihara, Y. Nakamura, S. Onishi, R. Shearer, D. Wendel, and L. Chang. A 4R2W register file for a 2.3GHz wire-speed POWER™ processor with double-pumped write operation. In 2011 IEEE International Solid-State Circuits Conference, pages 256-258, Feb 2011.

[13] H. E. Yantir, S. Bayar, and A. Yurdakul. Efficient implementations of multi-pumped multi-port register files in fpgas. In 2013 Euromicro Conference on Digital System Design, pages 185-192, Sept 2013.

[14] P. Meinerzhagen, C. Roth, and A. Burg. Towards generic low-power area-efficient stan- dard cell based memory architectures. In 2010 53rd IEEE International Midwest Sym- posium on Circuits and Systems, pages 129-132, Aug 2010.

[15] **H.** Zhang, X. Chen, N. Xiao, and F. Liu. Architecting energy-efficient STT-RAM based register file on gpgpus via delta compression. In *Proceedings of the 53rd Annual Design Automation Conference,* DAC '16, pages 119:1-119:6, New York, NY, USA, 2016. ACM.

[16] M. Horowitz, **P.** Chow, D. Stark, R. T. Simoni, A. Salz, S. Przybylski, J. Hennessy, G. Gulak, A. Agarwal, and J. M. Acken. MIPS-X: a 20-MIPS peak, 32-bit micropro- cessor with on-chip cache. *IEEE Journal of Solid-State Circuits,* 22(5):790-799, Oct 1987.

[17] N. Weste and D. Harris. CMOS VLSI Design: A Circuits and Systems Perspective. Addison Wesley, 2011.

[18] R. Kumar and G. Hinton. A family of 45nm IA processors. In 2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers, pages 58-59, Feb 2009.

[19] X. Liang and D. Brooks. Mitigating the impact of process variations on processor register files and execution units. In Proceedings of the 39th Annual IEEE/ ACM International Symposium on Microarchitecture, MICRO 39, pages 504-514, Washington, DC, USA, 2006. IEEE Computer Society.

[20] J. Chen, L. T. Clark, and T. H. Chen. An ultra-low-power memory with a subthreshold power supply voltage. IEEE Journal of Solid-State Circuits, 41(10):2344-2353, Oct 2006.

[21] N. Verma and A. P. Chandrakasan. A 256 kb 65 nm 8T subthreshold SRAM employing sense-amplifier redundancy. IEEE Journal of Solid-State Circuits, 43(1):141-149, Jan 2008.

[22] B. Stackhouse, S. Bhimji, C. Bostak, D. Bradley, B. Cherkauer, J. Desai, E. Francom,

[23] M. Gowan, P. Gronowski, D. Krueger, C. Morganti, and S. Troyer. A 65 nm 2-billion transistor quad-core itanium processor. IEEE Journal of Solid-State Circuits, 44(1):18- 31, Jan 2009.

[24] Y. Zhao, J. Li, and K. Mohanram. Multi-port FinFET SRAM design. In Proceedings of the 23rd ACM International Conference on Great Lakes Symposium on VLSI, GLSVLSI '13, pages 293-298, New York, NY, USA, 2013. ACM.

[25] H. Yan, Y. Liu, D. h. Wang, and C. h. Hou. A low-power 8-read 4-write register file design. In 2010 Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia), pages 178-181, Sept 2010.

[26] J. Pille, D. Wendel, 0. Wagner, R. Sautter, W. Penth, T. Froehnel, S. Buettner, 0. Tor- reiter, M. Eckert, J. Paredes, D. Hrusecky, D. Ray, and M. Canada. A 32kb 2R/1W 11 data cache in 45nm soi technology for the power7™ processor. In 2010 IEEE Interna- tional Solid-State Circuits Conference - (ISSCC), pages 344-345, Feb 2010.

[27] J. Warnock, D. Wendel, T. Aipperspach, E. Behnen, R. A. Cordes, S. H. Dhong, K. Hi- rairi, H. Murakami, S. Onishi, D. C. Pham, J. Pille, S. D. Posluszny, 0. Takahashi, and H. Wen. Circuit design techniques for a first-generation cell broadband engine processor. IEEE Journal of Solid-State Circuits, 41(8):1692-1706, Aug 2006.

[28] M. Golden, S. Hesley, A. Scherer, M. Crowley, S. C. Johnson, S. Meier, D. Meyer, J. D. Moench, S. Oberman, H. Partovi, F. Weber, S. White, T. Wood, and J. Yong. A seventh- generation x86 microprocessor. IEEE Journal of Solid-State Circuits, 34(11):1466-1477, Nov 1999.

[29] C. Zang, S. Imai, and S. Kimura. Duplicated register file design for embedded simulta- neous multithreading microprocessor. In 2005 6th International Conference on ASIC, volume 1, pages 90-93, Oct 2005.

[30] D. F. Wendel, R. Kalla, J. Warnock, R. Cargnoni, S. G. Chu, J. G. Clabes, D. Dreps, D. Hrusecky, J. Friedrich, S. Islam, J. Kahle, J. Leenstra, G. Mittal, J. Paredes, J. Pille, P. J. Restle, B. Sinharoy, G. Smith, W. J. Starke, S. Taylor, A. J. V. Norstrand, S. Weitzel, P. G. Williams, and V. Zyuban. Power7™ ;, a highly parallel, scalable multi-core high end server processor. IEEE Journal of Solid-State Circuits, 46(1):145- 161, Jan 2011.

[31] J. H. Tseng and K. Asanovic. Banked multiported register files for high-frequency superscalar microprocessors. In Proceedings of the 30th Annual International Symposium on Computer Architecture, ISCA '03, pages 62-71, New York, NY, USA, 2003. ACM.

[32] E. S. Fetzer, D. Dahle, C. Little, and K. Safford. The parity protected, multithreaded register files on the 90-nm itanium microprocessor. IEEE Journal of Solid-State Circuits, 41(1):246-255, Jan 2006.

[33] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. Cacti 5.1. Technical report, HP Laboratories, April 2008.